爱创课堂前端培训

JS 基础

第 7 天课堂笔记（本课程共 8 天）

班级：北京前端训练营 7 期

日期：2017 年 5 月 18 日

爱创课堂官网 ：www.icketang.com

目录

复习

字符串方法：

slice(start,end)截取区间字符串。

split()字符串切割父字符串返回值是数组。

substr(start,howmany)截取一定数目的字符串。

substring(start,end)自行判断数值大小，包括小的不包括大的。不能使用负值。

match()匹配，返回数组。通常使用全局匹配。

search()。查找返回值匹配的索引值。只能够返回第一次出现的位置。

charAt()返回指定索引值的字符。

indexOf()第一次出现的索引值。

toUpperCase()转换为大写。

toLowerCase()转换为小写。

concat()返回的新字符串。

replace()。替换

正则的方法:

exec()查找。

test()检测。返回值 true 或者 false

一、arguments

arguments 是一个类数组对象。

后台其他的语言都有一种函数重载现象。就是函数名相同，但是传递的参数不同。属于不同的函数。

js 没有重载现象。js 中函数名相同，传递的参数不同时，后面的会覆盖掉前边的。

```
function fun(a,b){
    console.log(1);
}
    function fun(a,b,c){
    console.log(2);
}
fun(1,2);
```

```
2
```

当函数执行时，传递的实际参数会存入到 arguments 这个类数组对象中。

arguments 可以通过下标获取任意一项。

```
sum(1,2,3,4,5,6,7)
console.log(arguments[0]);
console.log(arguments[1]);
console.log(arguments[2]);
console.log(arguments[3]);
console.log(arguments[4]);
console.log(arguments[5]);
console.log(arguments[6]);
console.log(arguments[7]);
console.log(arguments[8]);
console.log(arguments[9]);
```

```
1
2
3
4
5
6
7
undefined
undefined
undefined
```

arguments 还可以通过下标进行赋值

```
1  console.log(arguments[8] = 100);
```

```
100
```

还可以通过遍历得到所有项。

```
1  function sum(a,b){
       for(var i = 0 ; i <= arguments.length -1 ; i ++){
2          console.log(arguments[i]);
3      }
4
5  }
6  sum(1,2,3,4,5,6,7)
```

```
1
2
3
4
5
6
7
```

并不是所有的数组方法都适用于 arguments。

```
1  arguments.slice(2,5);
```

```
▶ [1, 2, 3, 4, 5, 6, 7]
▶ Uncaught TypeError: arguments.slice is not a function
    at sum (03_arguments.html:29)
    at 03_arguments.html:31
```

我们可以根据传递的实际参数的个数的不同，函数执行不同操作。（模拟函数的重载现象）。

案例：当函数传递 2 个参数时，求 2 个参数和；如果传递 3 个参数，比较前两个参数的大小取大值与第三个参数求和。

```
1  function sum(a,b,c){
2      if(arguments.length == 2){
```

```
        return a + b;
    }else{
        return (a > b ? a : b) + c;
    }
}

console.log(sum(1,2));
console.log(sum(1,2,3));
```

```
3
5
```

```
function sum(a,b,c){
    switch(arguments.length){
        case 2:
            return a + b;
            break;
        case 3:
            return (a > b ? a : b) + c;
            break;
        default:
            // alert("输入参数错误");
            // 抛出错误
            throw new Error("输入参数错误");
    }

}

console.log(sum(1,2));     //3
console.log(sum(1,2,3));   //5
console.log(sum(1,2,3,4)); //抛出错误
```

案例 2：当传递 1 个参数时，判断这个数的奇偶；如果传递 2 个参数，判断这两个参数的奇偶性是否相同。

```
function jiou(a,b){
    switch(arguments.length){
        // 一个参数时
        case 1:
            if(a % 2 == 0){
                alert(a + "是偶数")
            }else{
                alert(a + "是奇数")
            }
            break;
        // 2 个参数
        case 2:
            if((a + b) % 2 == 0){
                alert("奇偶性相同");
            }else{
                alert("奇偶性不同");
            }
            break;
        default:
            throw new Error("参数错误！");
    }
}
```

二、IIFE

IIFE 是一个缩写，immediately-invoked function expression。即时调用函数表达式。

IIFE 表示在函数定义的时候，就立即执行。

```
function fun(a,b){
    console.log(1);
}();
```

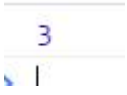
（）表示立即执行。但是不能直接跟在 function 定义函数的后面。（）只能跟在函数名或者函数表达式的后面。

```
// 把 function 关键字定义的函数矮化为表达式
// 在函数前面加数学运算符
// +function fun(a,b){
//  console.log(1);
// }();

// !function fun(a,b){
//  console.log(1);
// }();

//我们使用的是直接把函数的放小括号里面。
(function fun(a,b){
    console.log(1);
})();
```

实际工作中我们使用匿名函数的 IIFE。

```
(function(){})()
```

```
// 匿名函数的 IIFE
//实际参数书写在后面的小括号里面
(function(a,b){
    console.log(a + b);
})(1,2);
```



IIFE 能够关住自己的作用域.

```
// IIFE 能够关住自己的作用域
        (function fun(a,b){
            console.log(a + b);
        })();

 fun(1,2)
```

```
Uncaught ReferenceError: fun is not defined
  at 06_IIFE.html:53
```

```
(function(a){
            a++;
            console.log(a);
        })(5);

        (function(a){
            a++;
            console.log(a);
        })(8);
```

```
6
9
```

IIFE 中的匿名函数如果有返回值可以直接作为值参与运算。

```
// IIFE 可以作为值参与运算
        function sum(a,b,c){
            return (function(a,b){
                return (a > b ? a : b)
            })(a,b) + c;
        }

        console.log(sum(1,2,3));
```

三、结合数组观察闭包

数组中的数据可以是任何类型，现在让数组中的每一项存入一个函数。

```
var arr = [];
        for(var i = 0 ; i < 10 ;i ++){
            arr[i] = function(){
                console.log(i);
            }
        }
        console.log(arr);
        console.log(arr[1]());
        console.log(arr[2]());
        console.log(arr[3]());
```

```
10
undefined
10
undefined
10
undefined
```

```
// 使用 IIFE
        var arr = [];
        for(var i = 0 ; i < 10 ; i ++){
            (function(a){
```

```
5                 arr[a] = function(){
6                     console.log(a);
7                 }
8             })(i);
9         }
10
11        console.log(arr);
12        // console.log(arr[1]());
13        // console.log(arr[2]());
14        for(var i = 0 ; i < arr.length ; i ++){
15            console.log(arr[i]());
16        }
```

```
▶ [function, function, function, function, function, function, function, function, function, function]
0
undefined
1
undefined
2
undefined
3
undefined
4
undefined
5
undefined
6
undefined
7
undefined
8
undefined
9
```

四、DOM

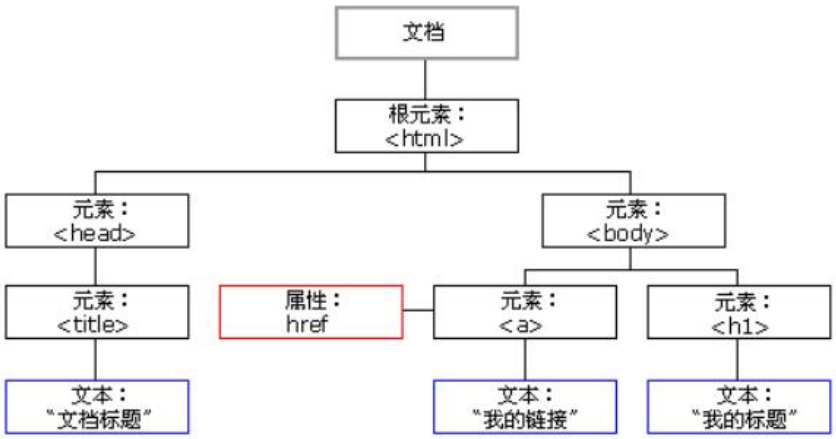
4.1 DOM概述

我们前面 JS 学习都是语言核心部分，也就是 ECMAscript。一般都是在控制台、输出语句里操作，JS 还包括 DOM 和 BOM。

DOM（Document Object Model，文档对象模型）描绘了一个层次化的节点树，允许开发人员添加、移除和修改页面的某一部分。

这使得 JavaScript 操作 HTML，不是在操作字符串，而是在操作节点，极大地降低了编程难度。

DOM 对很多东西做了抽象，提供了丰富的 API：取得元素、css 样式、事件、运动、元素尺寸位置、节点操作。



DOM节点树

体会 DOM 操作：

```
// 得到元素
var oBox = document.getElementById("box");
var oImg = document.getElementById("img");
//更改 css 样式
oBox.style.backgroundColor = "yellowgreen";
// 鼠标进入
oImg.onmouseover = function(){
    oImg.src = "images/baby.png";
}
// 鼠标移除
oImg.onmouseout = function(){
    oImg.src = "images/xiaoming.png";
}
```

4.2 html 操作

JavaScript 通过 document 对象表示文档，它表示整个页面。它有很多属性和方法，包含了绝大多数多页面的特征和操作。

DOM 操作，往往都是从取得某个（些）HTML 元素开始，然后对这个（些）元素进行一些操作。所以得到元素是非常重要的。得到元素的操作可以使用 document 对象的两个方法来完成：

我们常用的得到元素的方法：

```
document.getELementById()       //通过 id 名得到元素的方法
document.getElementsByTagName()//通过标签名得到元素的方法
```

通过 id 名获取元素：双引号中直接书写 id 名不要加＃。

```
var img = document.getElementById("#img"); //错误写法
```

通过 id 获取的元素是对象类型。可以直接使用点语法使用对象具有的属性和方法。

```
console.log(typeof img);
```

```
object
```

通过点语法获取元素的属性（读取）

```
// 通过点语法获取元素的属性
console.log(img.id);
console.log(img.src);
```

可以使用=进行赋值

```
// 可以使用=进行赋值
console.log(document.title);
document.title = "这是新的标题";
console.log(document.title);
```

其他的属性都可以进行赋值只有 id 不能。因为 id 是只读属性不能更改。

```
console.log(img.id);
```

```
img
```

自定义属性不能使用点语法操作。

```
console.log(box.hua);
```

```
undefined
```

自定义属性的读取和设置：

getAttribute()　读取

setAttribute()　设置

```
console.log(box.getAttribute("hua"));       //读取自定义属性
box.setAttribute("hua", "baicai"); //设置自定义属性
console.log(box.getAttribute("hua"));
```

```
youcai
baicai
```

点语法和 getAttribute（）,setAttribute()区别：

①点语法只能用于操作 html 原有的属性不能操作自定义属性。而 getAttribute（）,setAttribute()可以。

```
<div id="box" class="box" hua="youcai">内部文字</div>
console.log(box.hua);
console.log(box.getAttribute("hua"));
```

②点语法操作属性时，属性名可能需要更改。而 getAttribute（）,setAttribute()不需要。

class改为 className

for 改为 htmlFor

colspan 改为 colSpan

rowspan 改为 rowSpan。

```
console.log(box.className);
console.log(box.getAttribute("class"));
```

③点语法操作对象时得到的是对象，getAttribute（）得到的是字符串。

```
console.log(typeof box.style);
console.log(typeof box.getAttribute("style"));
```

```
object
string
```

④点操作：style 是对象可以继续打点。而 `getAttribute("style")`得到的是字符串不能进行一层操作。

点操作时用到复合属性需要使用驼峰命名法，而 `getAttribute("")`不需要。

```
console.log(box.style.backgroundColor);
console.log(box.style.borderRightColor);
```

总结：虽然点操作和 getAttribute（）,setAttribute()都可以读取属性或者设置属性，但是只有自定义属性才使用 getAttribute（）,setAttribute()。其他情况都使用点操作。

4.3 css 的操作

我们通过点语法得到的 style 是一个对象，继续通过打点调用属性。

通过 style 只能获取和设置行内的样式，不能得到计算后的样式。

通过获取属性重新等号赋值，等号右侧赋的新值需要用双引号包裹，原来 css 样式值怎么写，引号里就怎么写。

style 再下一层级的样式属性如果是复合属性，需要写驼峰式命名法。

```
box.style.backgroundColor = "pink";
box.style.borderWidth = "10px";
```

4.4 事件

事件监听：我们计算机在解析我们 JS 代码的时候，会去看某一些元素身上是否添加了事件。随时监听这些事件有没有被触发，如果触发就立即执行相应的行为。

onclick　　单击

ondblclick　　双击

onmouseover　鼠标移上

onmouseout	鼠标移出
onmousedown	鼠标按下
onmouseup	鼠标弹起
onfocus	获取焦点
onblur	失去焦点
onload	加载完毕之后

添加事件监听方法：通过给一个对象添加.事件，赋值一个函数。

这个函数会在事件被触发的时候立即执行。

```
box.onclick = fun;
function fun(){
    alert(box.innerHTML);
}
```

总结：执行函数又多了一种方法，可以将函数绑定给事件，触发事件就会立即执行函数。

函数名()，是我们之前学的方法。

js 书写位置一定要注意。如果写在 body 部分需要将 js 写在所有的 html 元素的最后。

```
window.onload = function(){
   这些语句是在 html 所有的元素加载完毕之后执行
}
```

onload，在我们的指定的元素加载完之后，执行。

```
<script type="text/javascript">
window.onload = function(){
}
</script>
```

下拉菜单的效果：

```
// 鼠标移进 box 出现 list
box.onmouseover = function(){
    list.style.display = "block";
}
// 鼠标移除 list 隐藏
box.onmouseout = function(){
    list.style.display = "none";
}
```

案例：字符串拼接的轮播图

```
img.src = "images/lunbo/" + index + ".jpg";
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```